SSH Keys

BV

[2015-09-20 Sun 12:00]

1 The problem

Over the years I've build up about a dozen SSH key pairs. They have tended to get used in two competing patterns one-to-many relationships:

1. One private key copied to many clients.
2. One public key copied to many servers.

This pattern is repeated for different domains. For example, the following keys exist:

- two used for decades for main interactive shell access at work and home with both public and private proliferated.
- one for every github identity maintained. Single public key placed on github, private key proliferated.
- various special purposes accounts with usually mild proliferation of both public and private halves.

This pattern of usage causes a practical problem that when too many private keys get loaded in the agent the matching key may not be tried before the connection negotiation fails (usually after 3 offered keys are unsuccessful).

This pattern also leads to a potential security vulnerability where the probability of a key compromise increases with the proliferation of the private key and the potential damage increases with the proliferation of the public key. Limiting key pairs to be associated with just one pair of connection end points maximizes security but also inconvenience. Servers that limit the number of public keys force one to either proliferate private keys or to forego connection except from one client.

2 Best compromise

Given the above, the best compromise (to avoid security compromise!) policy seems to be to prefer, in order of increase risk:

1. Generate a key pair unique to each client-server connection.
2. Where unacceptable, generate a single, default, unique key pair on every client account and proliferate it to those servers that support multiple public keys.
3. Where a server supports only a single authorized key, generate a single key pair and proliferate its private half to the clients that require it.

3 Key management

There is a matrix of N-clients, N-servers, N-public and N-private keys to manage. Public keys may be distributed to all servers but only select ones shall be placed in `~/.authorized_keys` and on a per-server basis. On the other hand, private keys should only be distributed to select clients as required and no more. To avoid the "too many keys" problem described above, a configuration needs to be maintained to direct select private keys to be used to connect to select servers. This configuration has mild security sensitivity as it implies which server authorized which public key. Its use can also be location dependent as some host names may only be resolved depending on the available DNS servers (eg, home router vs work vs hotel).

3.1 Layout of `~/.ssh/`

The parts of `~/.ssh/` that are safe to distribute (all but private keys) will be kept in git and distributed.

```
.ssh/
  authorized_keys.d/pubkeys/<*.pub>       # distributed by default
  authorized_keys.d/<server>/<symlinks>   # distributed by default
  config.d/fragments/                     # distributed by default
  config.d/<client>/<symlnks>             # distributed by default
  privkeys/                               # not distributed by default
```

the `pubkeys/` directory holds all public key files. The `authorized_keys.d/<server>/` directory holds symbolic links to files in `pubkeys/` which are authorized to

log in. Likewise, `config.d/fragments` holds configuration fragments which get linked into a client-specific sub directory. To configure a client or server account one does:

```
(server)$ cat ~/.ssh/authorized_keys.d/‘hostname‘/* > ~/.ssh/authorized_keys
(client)$ cat ~/.ssh/config.d/‘hostname‘/* > ~/.ssh/config
```

4 Configuration Management

The three policies are supported by `~/.ssh/config` patterns.

For the first policy, where a single, unique key pair is used for exactly one client to connect to one server a configuration stanza like the following is used:

```
Host ...
  User ...
  Hostname ...
  IdentityFile ~/.ssh/privkeys/id_%u-%l-%r-%h
```

When the default key for the user is acceptable then the identity file is used:

```
Host ...
  IdentityFile ~/.ssh/privkeys/id_%u-%l
```

When a key is distributed to many clients for accessing a single server then:

```
Host ...
  IdentityFile ~/.ssh/privkeys/id_%r-%h
```

Private keys take similar names but with `.pub` appended.